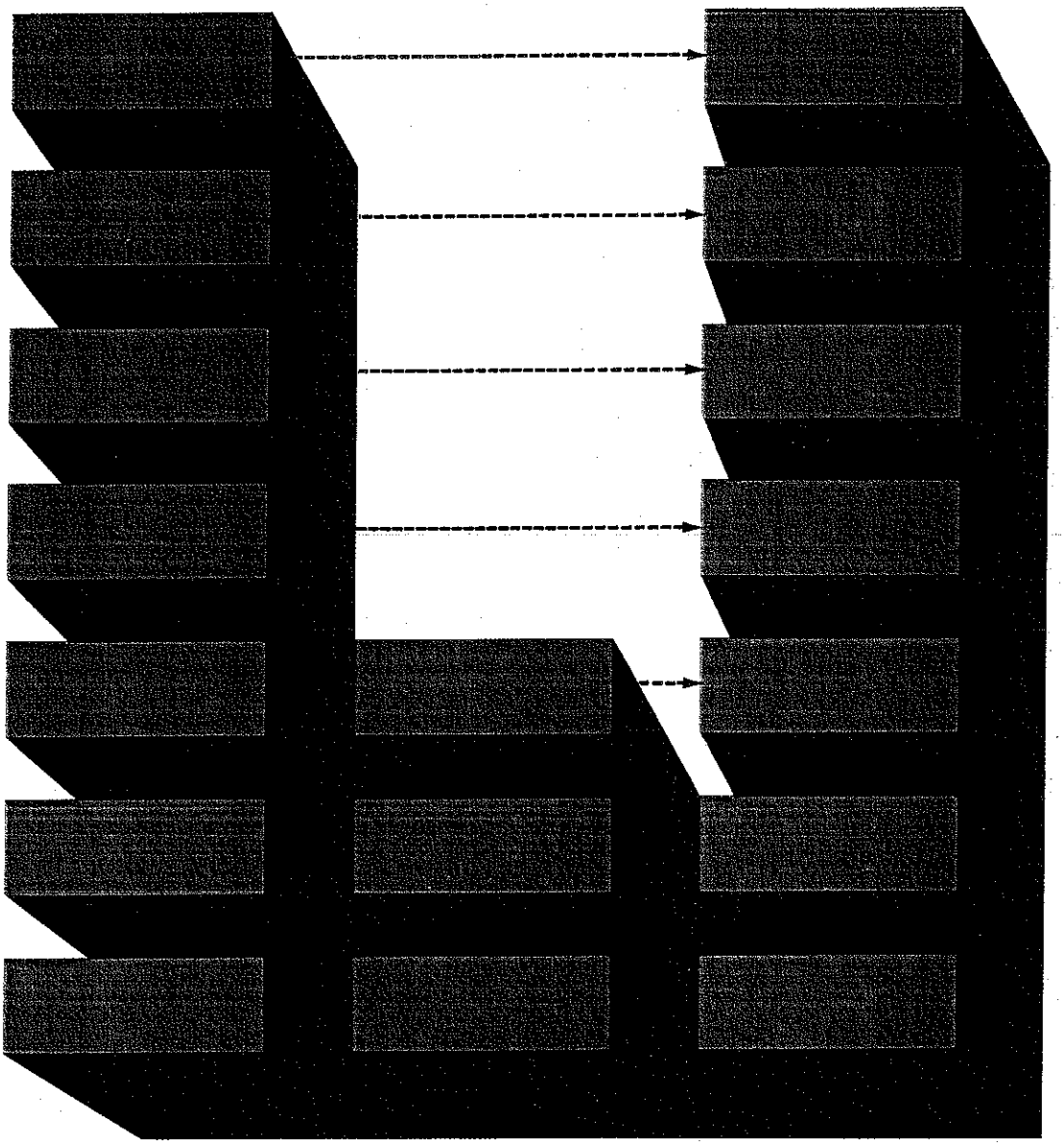
Exhibit 6

SECOND EDITION

COMPUTER NETWORKS



ANDREW S. TANENBAUM

COMPUTER NETWORKS

SECOND EDITION

ANDREW S. TANENBAUM

Vrije Universiteit
Amsterdam, The Netherlands

PRENTICE HALL

ENGLEWOOD CLIFFS, NEW JERSEY 07632

Library of Congress Cataloging-in-Publication Data
Tanenbaum, Andrew S.
Computer networks / Andrew S. Tanenbaum.—2nd ed.
p. cm.
Bibliography: p.
Includes index.
ISBN 0-13-162959-X
1. Computer networks. I. Title.
TK5105.5.T36 1988
004.6—dc19 88-16937
CIP

Editorial/production supervision: **Lisa Schulz Garboski**
Manufacturing buyer: **Mary Ann Gloriande**

Prentice Hall Software Series
Brian W. Kernighan, Advisor

The publisher offers discounts on this book when ordered in bulk quantities. For more information, write:

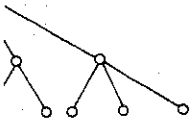
Special Sales/College Marketing
Prentice Hall
College Technical and Reference Division
Englewood Cliffs, NJ 07632

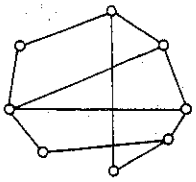
Printed in the United States of America

10 9

ISBN 0-13-162959-X

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*





(f)

(b) Ring.

ıg. The remain-
1 groups. Each
)acket with, say,
:s subscribing to

ets. In a bus or
wed to transmit.
ıtion mechanism
) transmit simul-
uted.
ach IMP has an
the output *from*
transmissions of

ıtes around on its
/pically, each bit
ı few bits, often
er broadcast sys-
he ring. Various

ic, depending on

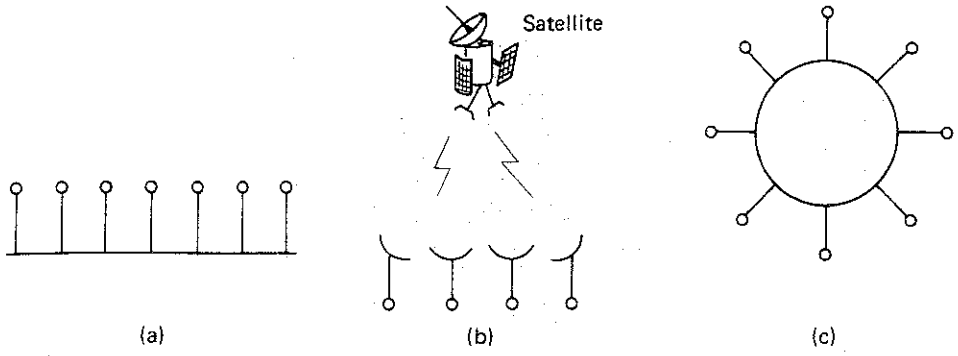


Fig. 1-4. Communication subnets using broadcasting. (a) Bus. (b) Satellite or radio. (c) Ring.

how the channel is allocated. A typical static allocation would be to divide up time into discrete intervals, and run a round robin, allowing each machine to broadcast only when its time slot comes up. Static allocation wastes channel capacity when a machine has nothing to say during its allocated slot, so some systems attempt to allocate the channel dynamically (i.e., on demand).

Dynamic allocation methods for a common channel are either centralized or decentralized. In the centralized channel allocation method, there is a single entity, for example a bus arbitration unit, which determines who goes next. It might do this by accepting requests and making a decision according to some internal algorithm. In the decentralized channel allocation method, there is no central entity; each machine must decide for itself whether or not to transmit. You might think that this always leads to chaos, but it does not. Later we will study many algorithms designed to bring order out of the potential chaos.

1.3. NETWORK ARCHITECTURES

Modern computer networks are designed in a highly structured way. In the following sections we examine the structuring technique in some detail.

1.3.1. Protocol Hierarchies

To reduce their design complexity, most networks are organized as a series of **layers** or **levels**, each one built upon its predecessor. The number of layers, the name of each layer, the contents of each layer, and the function of each layer differ from network to network. However, in all networks, the purpose of each layer is to offer certain services to the higher layers, shielding those layers from the details of how the offered services are actually implemented.

Layer n on one machine carries on a conversation with layer n on another machine. The rules and conventions used in this conversation are collectively known as the layer n **protocol**, as illustrated in Fig. 1-5 for a seven-layer network. The entities comprising the corresponding layers on different machines are called **peer processes**. In other words, it is the peer processes that communicate using the protocol.

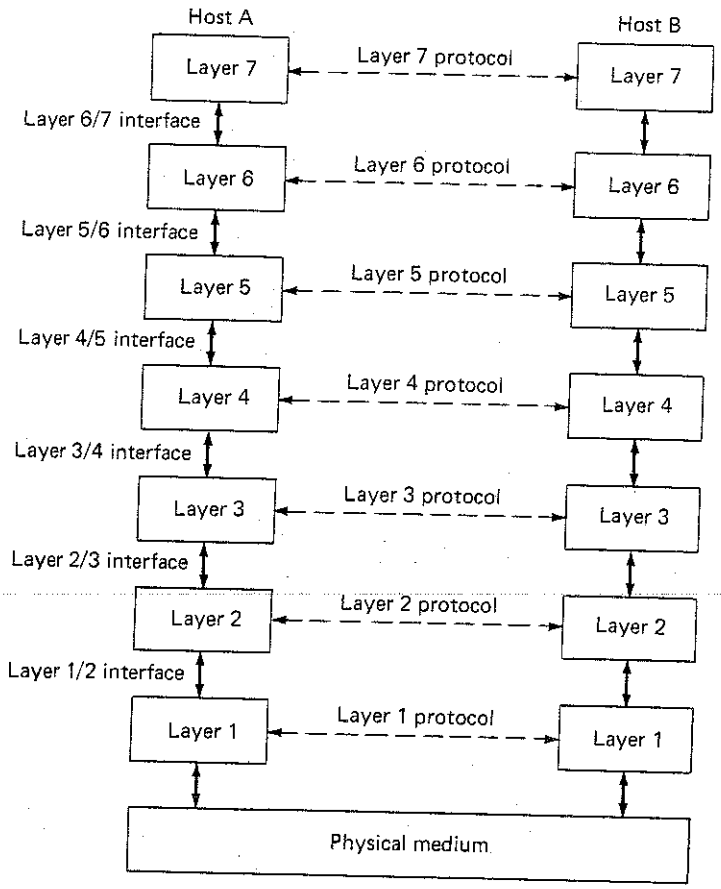


Fig. 1-5. Layers, protocols, and interfaces.

In reality, no data are directly transferred from layer n on one machine to layer n on another machine. Instead, each layer passes data and control information to the layer immediately below it, until the lowest layer is reached. Below layer 1 is the **physical medium** through which actual communication occurs. In Fig. 1-5, virtual communication is shown by dotted lines and physical communication by solid lines.

Between each pair of adjacent layers there is an **interface**. The interface defines which primitive operations and services the lower layer offers to the upper one. When network designers decide how many layers to include in a network and

what each one should do, one of the most important considerations is defining clean interfaces between the layers. Doing so, in turn, requires that each layer perform a specific collection of well-understood functions. In addition to minimizing the amount of information that must be passed between layers, clean-cut interfaces also make it simpler to replace the implementation of one layer with a completely different implementation (e.g., all the telephone lines are replaced by satellite channels), because all that is required of the new implementation is that it offer exactly the same set of services to its upstairs neighbor as the old implementation did.

The set of layers and protocols is called the **network architecture**. The specification of the architecture must contain enough information to allow an implementer to write the program or build the hardware for each layer so that it will correctly obey the appropriate protocol. Neither the details of the implementation nor the specification of the interfaces are part of the architecture because these are hidden away inside the machines and not visible from the outside. It is not even necessary that the interfaces on all machines in a network be the same, provided that each machine can correctly use all the protocols. The subjects of network architectures and protocols are the principal topics of this book.

An analogy may help explain the idea of multilayer communication. Imagine two philosophers (peer processes in layer 3), one in Kenya and one in Indonesia, who want to communicate. Since they have no common language, they each engage a translator (peer processes at layer 2), each of whom in turn contacts an engineer (peer processes in layer 1). Philosopher 1 wishes to convey his affection for *oryctolagus cuniculus* to his peer. To do so, he passes a message (in Swahili) across the 2/3 interface, to his translator, who might render it as "I like rabbits" or "J'aime des lapins" or "Ik hou van konijnen," depending on the layer 2 protocol.

The translator then gives the message to his engineer for transmission, by telegram, telephone, computer network, or some other means, depending on what the two engineers have agreed on in advance (the layer 1 protocol). When the message arrives, it is translated into Indonesian and passed across the 2/3 interface to philosopher 2. Note that each protocol is completely independent of the other ones as long as the interfaces are not changed. The translators can switch from French to Dutch at will, provided that they both agree, and neither changes his interface with either layer 1 or layer 3.

Now consider a more technical example: how to provide communication to the top layer of the seven-layer network in Fig. 1-6. A message, m, is produced by a process running in layer 7. The message is passed from layer 7 to layer 6 according to the definition of the layer 6/7 interface. In this example, layer 6 transforms the message in certain ways (e.g., text compression), and then passes the new message, M, to layer 5 across the layer 5/6 interface. Layer 5, in the example, does not modify the message but simply regulates the direction of flow (i.e., prevents an incoming message from being handed to layer 6 while layer 6 is busy handing a series of outgoing messages to layer 5).

In many networks, there is no limit to the size of messages accepted by layer 4,

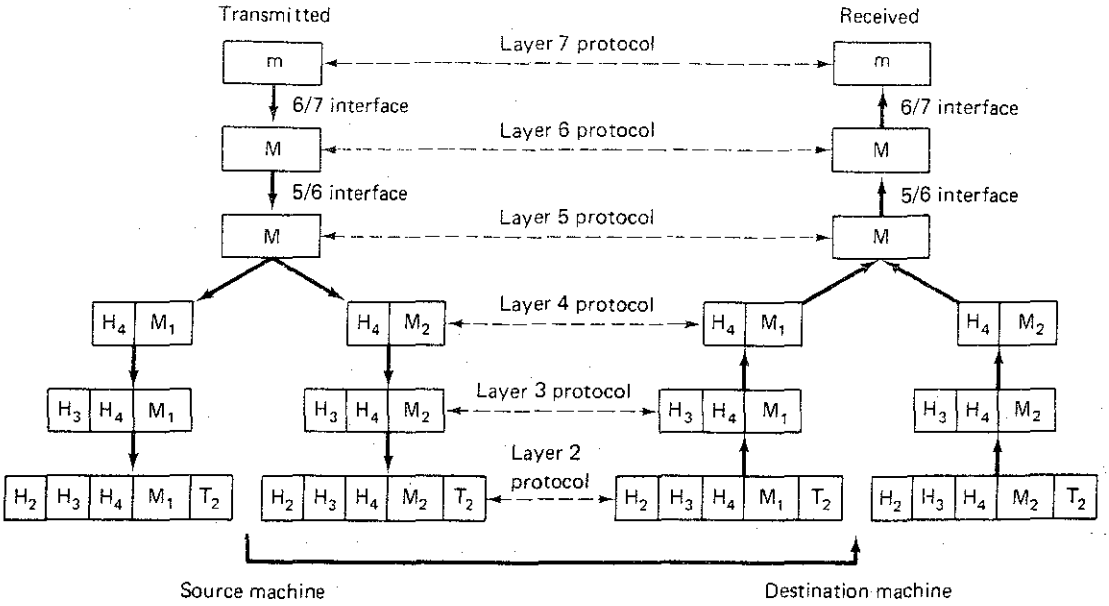


Fig. 1-6. Example information flow supporting virtual communication in layer 7.

but there is a limit imposed by layer 3. Consequently, layer 4 must break up the incoming messages into smaller units, prepending a **header** to each unit. The header includes control information, such as sequence numbers, to allow layer 4 on the destination machine to get the pieces back together in the right order if the lower layers do not maintain sequence. In many layers, headers also contain contain sizes, times and other control fields.

Layer 3 decides which of the outgoing lines to use, attaches its own headers, and passes the data to layer 2. Layer 2 adds not only a header to each piece, but also a trailer, and gives the resulting unit to layer 1 for physical transmission. At the receiving machine the message moves upward, from layer to layer, with headers being stripped off as it progresses. None of the headers for layers below n are passed up to layer n.

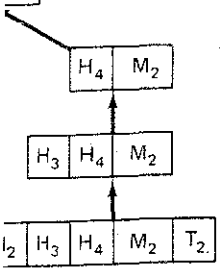
The important thing to understand about Fig. 1-6 is the relation between the virtual and actual communication and the difference between protocols and interfaces. The peer processes in layer 4, for example, conceptually think of their communication as being "horizontal," using the layer 4 protocol. Each one is likely to have a procedure called *SendToOtherSide* and a procedure *GetFromOtherSide*, even though these procedures actually communicate with lower layers across the 3/4 interface, not with the other side.

The peer process abstraction is crucial to all network design. Without this abstraction technique, it would be difficult, if not impossible, to partition the design of the complete network, an unmanageable problem, into several smaller, manageable, design problems, namely the design of the individual layers.

ed

6/7 interface

5/6 interface



achine

in layer 7.

ıust break up the
each unit. The
allow layer 4 on
right order if the
also contain con-

its own headers,
o each piece, but
transmission. At
ayer, with headers
yers below *n* are

n between the vir-
ols and interfaces.
their communica-
is likely to have a
nOtherSide, even
ers across the 3/4

ign. Without this
artition the design
smaller, manage-

1.3.2. Design Issues for the Layers

Some of the key design issues that occur in computer networking are present in several layers. Below, we will briefly mention some of the more important ones.

Every layer must have a mechanism for connection establishment. Since a network normally has many computers, some of which have multiple processes, a means is needed for a process on one machine to specify with whom it wants to establish a connection. As a consequence of having multiple destinations, some form of addressing is needed in order to specify a specific destination.

Closely related to the mechanism for establishing connections across the network is the mechanism for terminating them once they are no longer needed. As we will see in Chapter 6, this seemingly trivial point can actually be quite subtle.

Another set of design decisions concerns the rules for data transfer. In some systems, data only travel in one direction (**simplex communication**). In others they can travel in either direction, but not simultaneously (**half-duplex communication**). In still others they travel in both directions at once (**full-duplex communication**). The protocol must also determine how many logical channels the connection corresponds to, and what their priorities are. Many networks provide at least two logical channels per connection, one for normal data and one for urgent data.

Error control is an important issue because physical communication circuits are not perfect. Many error-detecting and error-correcting codes are known, but both ends of the connection must agree on which one is being used. In addition, the receiver must have some way of telling the sender which messages have been correctly received and which have not.

Not all communication channels preserve the order of messages sent on them. To deal with a possible loss of sequencing, the protocol must make explicit provision for the receiver to allow the pieces to be put back together properly. An obvious solution is to number the pieces, but this solution still leaves open the question of what should be done with pieces that arrive out of order.

An issue that occurs at every level is how to keep a fast sender from swamping a slow receiver with data. Various solutions have been proposed and will be discussed later. All of them involve some kind of feedback from the receiver to the sender, either directly or indirectly, about the receiver's current situation.

Another problem that must be solved at several levels is the inability of all processes to accept arbitrarily long messages. This property leads to mechanisms for disassembling, transmitting, and then reassembling messages. A related issue is what to do when processes insist upon transmitting data in units that are so small that sending each one separately is inefficient. Here the solution is to gather together several small messages heading toward a common destination into a single large message, and dismember the large message at the other side.

When it is inconvenient or expensive to set up a separate connection for each pair of communicating processes, the underlying layer may decide to use the same connection for multiple, unrelated conversations. As long as this multiplexing and

demultiplexing is done transparently, it can be used by any layer. Multiplexing is needed in the physical layer, for example, where all the traffic for all connections has to be sent over at most a few physical circuits.

When there are multiple paths between source and destination, a route must be chosen. Sometimes this decision must be split over two or more layers. For example, to send data from London to Rome, a high level decision might have to be made to go via France or Germany based on their respective privacy laws, and a low-level decision might have to be made to choose one of the many available circuits based on current traffic.

1.4. THE OSI REFERENCE MODEL

Now that we have discussed layered networks in the abstract, it is time to look at the set of layers that will be used throughout this book. The model is shown in Fig. 1-7. This model is based on a proposal developed by the International Standards Organization (ISO) as a first step toward international standardization of the various protocols (Day and Zimmermann, 1983). The model is called the **ISO OSI Open Systems Interconnection) Reference Model** because it deals with connecting open systems—that is, systems that are open for communication with other systems. We will usually just call it the OSI model for short.

The OSI model has seven layers. The principles that were applied to arrive at the seven layers are as follows:

1. A layer should be created where a different level of abstraction is needed.
2. Each layer should perform a well defined function.
3. The function of each layer should be chosen with an eye toward defining internationally standardized protocols.
4. The layer boundaries should be chosen to minimize the information flow across the interfaces.
5. The number of layers should be large enough that distinct functions need not be thrown together in the same layer out of necessity, and small enough that the architecture does not become unwieldy.

In Sections 1.5.1 through 1.5.7 we will discuss each layer of the model in turn, starting at the bottom layer. Note that the OSI model itself is not a network architecture because it does not specify the exact services and protocols to be used in each layer. It just tells what each layer should do. However, ISO has also produced standards for all the layers, although these are not strictly speaking part of the model. Each one has been published as a separate international standard.

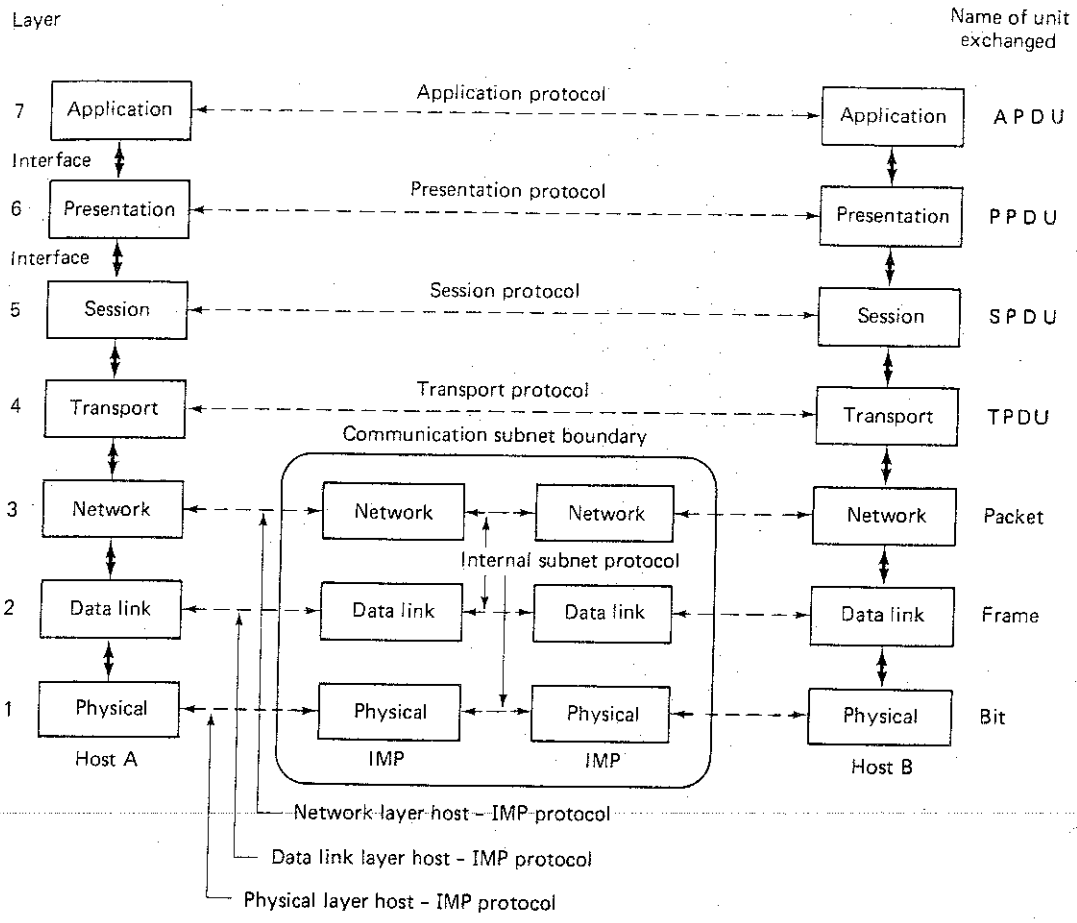


Fig. 1-7. The network architecture used in this book. It is based on the OSI model.

1.4.1. The Physical Layer

The **physical layer** is concerned with transmitting raw bits over a communication channel. The design issues have to do with making sure that when one side sends a 1 bit, it is received by the other side as a 1 bit, not as a 0 bit. Typical questions here are how many volts should be used to represent a 1 and how many for a 0, how many microseconds a bit lasts, whether transmission may proceed simultaneously in both directions, how the initial connection is established and how it is torn down when both sides are finished, and how many pins the network connector has and what each pin is used for. The design issues here largely deal with mechanical, electrical, and procedural interfaces, and the physical transmission medium, which lies below the physical layer. Physical layer design can be properly considered to be within the domain of the electrical engineer.

1.4.2. The Data Link Layer

The main task of the **data link layer** is to take a raw transmission facility and transform it into a line that appears free of transmission errors to the network layer. It accomplishes this task by having the sender break the input data up into **data frames** (typically a few hundred bytes), transmit the frames sequentially, and process the **acknowledgement frames** sent back by the receiver. Since the physical layer merely accepts and transmits a stream of bits without any regard to meaning or structure, it is up to the data link layer to create and recognize frame boundaries. This can be accomplished by attaching special bit patterns to the beginning and end of the frame. If these bit patterns can accidentally occur in the data, special care must be taken to avoid confusion.

A noise burst on the line can destroy a frame completely. In this case, the data link layer software on the source machine must retransmit the frame. However, multiple transmissions of the same frame introduce the possibility of duplicate frames. A duplicate frame could be sent, for example, if the acknowledgement frame from the receiver back to the sender was destroyed. It is up to this layer to solve the problems caused by damaged, lost, and duplicate frames. The data link layer may offer several different service classes to the network layer, each of a different quality and with a different price.

Another issue that arises in the data link layer (and most of the higher layers as well) is how to keep a fast transmitter from drowning a slow receiver in data. Some traffic regulation mechanism must be employed to let the transmitter know how much buffer space the receiver has at the moment. Frequently, this flow regulation and the error handling are integrated, for convenience.

If the line can be used to transmit data in both directions, this introduces a new complication that the data link layer software must deal with. The problem is that the acknowledgement frames for A to B traffic compete for the use of the line with data frames for the B to A traffic. A clever solution (piggybacking) has been devised; we will discuss it in detail later.

1.4.3. The Network Layer

The **network layer** is concerned with controlling the operation of the subnet. A key design issue is determining how packets are routed from source to destination. Routes could be based on static tables that are "wired into" the network and rarely changed. They could also be determined at the start of each conversation, for example a terminal session. Finally, they could be highly dynamic, being determined anew for each packet, to reflect the current network load.

If too many packets are present in the subnet at the same time, they will get in each other's way, forming bottlenecks. The control of such congestion also belongs to the network layer.

Since the operators of the subnet may well expect remuneration for their efforts,

there is often some accounting function built into the network layer. At the very least, the software must count how many packets or characters or bits are sent by each customer, to produce billing information. When a packet crosses a national border, with different rates on each side, the accounting can become complicated.

When a packet has to travel from one network to another to get to its destination, many problems can arise. The addressing used by the second network may be different from the first one. The second one may not accept the packet at all because it is too large. The protocols may differ, and so on. It is up to the network layer to overcome all these problems to allow heterogenous networks to be interconnected.

In broadcast networks, the routing problem is simple, so the network layer is often thin or even nonexistent.

1.4.4. The Transport Layer

The basic function of the **transport layer**, is to accept data from the session layer, split it up into smaller units if need be, pass these to the network layer, and ensure that the pieces all arrive correctly at the other end. Furthermore, all this must be done efficiently, and in a way that isolates the session layer from the inevitable changes in the hardware technology.

Under normal conditions, the transport layer creates a distinct network connection for each transport connection required by the session layer. If the transport connection requires a high throughput, however, the transport layer might create multiple network connections, dividing the data among the network connections to improve throughput. On the other hand, if creating or maintaining a network connection is expensive, the transport layer might multiplex several transport connections onto the same network connection to reduce the cost. In all cases, the transport layer is required to make the multiplexing transparent to the session layer.

The transport layer also determines what type of service to provide the session layer, and ultimately, the users of the network. The most popular type of transport connection is an error-free point-to-point channel that delivers messages in the order in which they were sent. However, other possible kinds of transport service are transport of isolated messages with no guarantee about the order of delivery, and broadcasting of messages to multiple destinations. The type of service is determined when the connection is established.

The transport layer is a true source-to-destination or **end-to-end** layer. In other words, a program on the source machine carries on a conversation with a similar program on the destination machine, using the message headers and control messages. In the lower layers, the protocols are between each machine and its immediate neighbors, and not by the ultimate source and destination machines, which may be separated by many IMPs. The difference between layers 1 through 3, which are chained, and layers 4 through 7, which are end-to-end, is illustrated in Fig. 1-7.

Many hosts are multiprogrammed, which implies that multiple connections will be entering and leaving each host. There needs to be some way to tell which message belongs to which connection. The transport header (H_4 in Fig. 1-6) is one place this information could be put.

In addition to multiplexing several message streams onto one channel, the transport layer must take care of establishing and deleting connections across the network. This requires some kind of naming mechanism, so that a process on one machine has a way of describing with whom it wishes to converse. There must also be a mechanism to regulate the flow of information, so that a fast host cannot overrun a slow one. Flow control between hosts is distinct from flow control between IMPs, although we will later see that similar principles apply to both.

1.4.5. The Session Layer

The session layer allows users on different machines to establish **sessions** between them. A session allows ordinary data transport, as does the transport layer, but it also provides some enhanced services useful in a some applications. A session might be used to allow a user to log into a remote time-sharing system or to transfer a file between two machines.

One of the services of the session layer is to manage dialogue control. Sessions can allow traffic to go in both directions at the same time, or in only one direction at a time. If traffic can only go one way at a time (analogous to a single railroad track), the session layer can help keep track of whose turn it is.

A related session service is **token management**. For some protocols, it is essential that both sides do not attempt the same operation at the same time. To manage these activities, the session layer provides tokens that can be exchanged. Only the side holding the token may perform the critical operation.

Another session service is **synchronization**. Consider the problems that might occur when trying to do a two-hour file transfer between two machines on a network with a 1 hour mean time between crashes. After each transfer was aborted, the whole transfer would have to start over again, and would probably fail again when the network next crashed. To eliminate this problem, the session layer provides a way to insert checkpoints into the data stream, so that after a crash, only the data after the last checkpoint have to be repeated.

1.4.6. The Presentation Layer

The **presentation layer** performs certain functions that are requested sufficiently often to warrant finding a general solution for them, rather than letting each user solve the problems. In particular, unlike all the lower layers, which are just interested in moving bits reliably from here to there, the presentation layer is concerned with the syntax and semantics of the information transmitted.

A typical example of a presentation service is encoding data in a standard agreed upon way. Most user programs do not exchange random binary bit strings. They exchange things such as people's names, dates, amounts of money, and invoices. These items are represented as character strings, integers, floating point numbers, and data structures composed of several simpler items. Different computers have different codes for representing character strings (e.g., ASCII and EBCDIC), integers (e.g., one's complement and two's complement), and so on. In order to make it possible for computers with different representations to communicate, the data structures to be exchanged can be defined in an abstract way, along with a standard encoding to be used "on the wire." The job of managing these abstract data structures and converting from the representation used inside the computer to the network standard representation is handled by the presentation layer.

The presentation layer is also concerned with other aspects of information representation. For example, data compression can be used here to reduce the number of bits that have to be transmitted and cryptography is frequently required for privacy and authentication.

1.4.7. The Application Layer

The application layer contains a variety of protocols that are commonly needed. For example, there are hundreds of incompatible terminal types in the world. Consider the plight of a full screen editor that is supposed to work over a network with many different terminal types, each with different screen layouts, escape sequences for inserting and deleting text, moving the cursor, etc.

One way to solve this problem is to define an abstract **network virtual terminal** that editors and other programs can be written to deal with. To handle each terminal type, a piece of software must be written to map the functions of the network virtual terminal onto the real terminal. For example, when the editor moves the virtual terminal's cursor to the upper left-hand corner of the screen, this software must issue the proper command sequence to the real terminal to get its cursor there too. All the virtual terminal software is in the application layer.

Another application layer function is file transfer. Different file systems have different file naming conventions, different ways of representing text lines, and so on. Transferring a file between two different systems requires handling these and other incompatibilities. This work, too, belongs to the application layer, as do electronic mail, remote job entry, directory lookup, and various other general-purpose and special-purpose facilities.

1.4.8. Data Transmission in the OSI Model

Figure 1-8 shows an example of how data can be transmitted using the OSI model. The sending process has some data it wants to send to the receiving process. It gives the data to the application layer, which then attaches the application

header, *AH* (which may be null), to the front of it and gives the resulting item to the presentation layer.

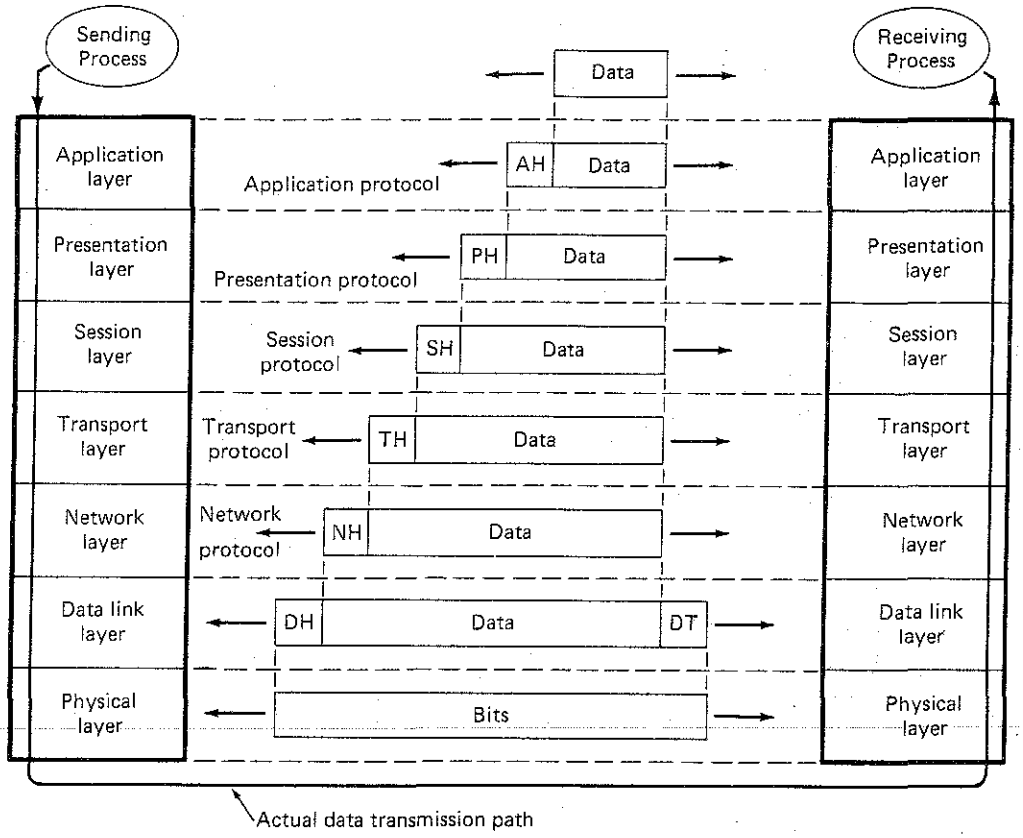


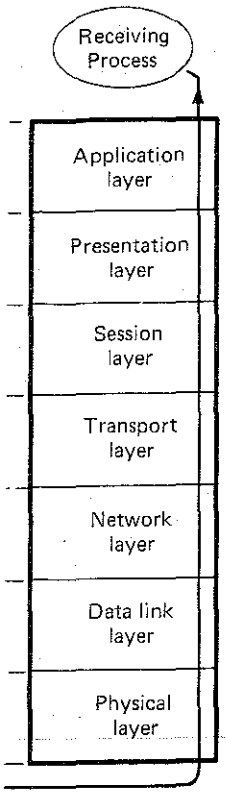
Fig. 1-8. An example of how the OSI model is used. Some of the headers may be null. (Source: H.C. Folts. Used with permission.)

The presentation layer may transform this item in various ways, and possibly add a header to the front, giving the result to the session layer. It is important to realize that the presentation layer is not aware of which portion of the data given to it by the application layer is *AH*, if any, and which is true user data. Nor should it be aware.

This process is repeated until the data reach the physical layer, where they are actually transmitted to the receiving machine. On that machine the various headers are stripped off one by one as the message propagates up the layers until it finally arrives at the receiving process.

The key idea throughout is that although actual data transmission is vertical in Fig. 1-8, each layer is programmed as though it were really horizontal. When the sending transport layer, for example, gets a message from the session layer, it attaches a transport header and sends it to the receiving transport layer. From its

ulting item to the



ıders may be

'ays, and possibly
It is important to
f the data given to
ata. Nor should it

:r, where they are
ıe various headers
'ers until it finally

ssion is vertical in
zontal. When the
session layer, it
rt layer. From its

point of view, the fact that it must actually hand the message to the network layer on its own machine is an unimportant technicality. As an analogy, when an Uighur-speaking diplomat is addressing the United Nations, he thinks of himself as addressing the other assembled diplomats. That, in fact, he is really only speaking to his translator is seen as a technical detail.

1.5. SERVICES

The real function of each layer in the OSI model is to provide services to the layer above it. In this section we will look at precisely what a service is in more detail, but first we will give some of the OSI terminology.

1.5.1. OSI Terminology

The active elements in each layer are called **entities**. An entity can be a software entity (such as a process), or a hardware entity (such as an intelligent I/O chip). Entities in the same layer on different machines are called **peer entities**. The layer 7 entities are called **application entities**; the layer 6 entities are called **presentation entities**, and so on.

The entities in layer N implement a service used by layer $N + 1$. In this case layer N is called the **service provider** and layer $N + 1$ is called the **service user**. Layer N may use the services of layer $N - 1$ in order to provide its service. It may offer several classes of service, for example, fast, expensive communication and slow, cheap communication.

Services are available at **SAP**s. (**service access points**), The layer N SAPs are the places where layer $N + 1$ can access the services offered. Each SAP has an address that uniquely identifies it. To make this point clearer, the SAPs in the telephone system are the sockets into which modular telephones can be plugged, and the SAP addresses are the telephone numbers of these sockets. To call someone, you must know his SAP address. Similarly, in the postal system, the SAP addresses are street addresses and post office boxes. To send a letter, you must know the addressee's SAP address. In Berkeley UNIX†, the SAPs are the sockets and the SAP addresses are the socket numbers. The SAP concept is discussed in detail by Tomas et al. (1987).

In order for two layers to exchange information, there has to be an agreed upon set of rules about the **interface**. At a typical interface, the layer $N + 1$ entity passes an **IDU** (**Interface Data Unit**) to the layer N entity through the SAP as shown in Fig. 1-9. The IDU consists of an **SDU** (**Service Data Unit**) and some control information. The SDU is the information passed across the network to the peer entity

† UNIX is a registered trademark of AT&T Bell Laboratories.